МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: БИНАРНЫЕ ДЕРЕВЬЯ

Вариант № 6-в

Студент гр. 8304 Рыжиков А. В.
Преподаватель Фирсов К. В.

Санкт-Петербург
2019

1 Цель работы. Изучить бинарные деревья и леса, реализовать бинарное дерево на векторе.

Для заданного леса с произвольным типом элементов:
-получить естественное представление леса бинарным деревом
-вывести лес и бинарное дерево
-перечислить элементы леса в горизонтальном порядке(в ширину)

2 Описание программы

Программа считывает строку, содержащую лес, который представлен в скобочной записи. (разные деревья отделяются знаком ‘,’). Программа выводит элементы леса в горизонтальном порядке(в ширину). После программа обрабатывает лес и создаёт по нему бинарное дерево. Алгоритм построен на основе вектора.

Описание функций программы

1. `bool checkExpression(int &a, const std::string my_string)`
   Функция проверяет выражение на наличие ошибок в записи, количество символов, правильную последовательность, допустимые символы.
2. `void  treeDataProcessing (std::string &my_string)`
   Функция обрабатывает данные дерева, заполняет массив данными.
3. `int findParent(int num, int your_deep)`
   Функция находит родителя для нового элемента. Это крайний левый элемент в массиве у которого на 1 меньше значение

глубины. Если такой элемент не найден, то возвращается -1 для корня.

4. `void setNewDeep(int index)`

   Функция рекурсивно устанавливает новую глубину для элемента и всех его детей.

5. `void createBinaryTree()`

   Функция создаёт бинарное дерево по данным, которые содержатся в массиве.

6. `printInWidth()`

   Функция выводит элементы леса в горизонтальном порядке(в ширину).

7. `void printlnTree()`

   Функция выводит получившееся бинарное дерево и отладучную информация о глубине и родителе каждого элемента.

8. `printResult(int parent)`

   Функция выводит получившееся бинарное дерево .

Тесты

Тесты удовлетворяющие понятию формула

(a)

Output in width: a

Output binareThree: (a)

(a(b))

Output in width: ab

Output binareThree: (a(b))

(a),(b)

Output in width: ab

Output binareThree: (a(b))

(a(b)(c)(d))

Output in width: abcd

Output binareThree: (a(b(c(d))))

(a(b(c)(e(f))))

Output in width: abcef

Output binareThree: (a(b(c(e(f)))))

(a(b(c)(d)(e))),(f(i)(k)(l(m)(n)))

Output in width: afbiklcdemn

Output binareThree: (a(b(c(d(e))))(f(i(k(l(m(n)))))))

(a(a)(a)(a)(a)(a)(a)(a)(a)(a)(a)(a)(a)(a)(a)(a))

Output in width: aaaaaaaaaaaaaaa

Output binareThree: (a(a(a(a(a(a(a(a(a(a(a(a(a(a(a)))))))))))))))

(a(a)(a)(a)(a)(a)(a)),(b(b(b(b(b)))))

Output in width: abaaaaaabbbb

Output binareThree: (a(a(a(a(a(a))))))(b(b(b(b(b))))))

(a(e)(f)),(b(g(k)(l))),(c(n(r)))

Output in width: abcefgnklr

Output binareThree: (a(e(f))(b(g(k(l)))(c(n(r)))))

(a(b)(c(e)(f))(d))

Output in width: abcdef

Output binareThree: (a(b(c(e(f))(d))))

(g(k(m)(n))(l)),(o)

Output in width: goklmn

Output binareThree: (g(k(m(n))(l))(o))

(a(b)(c(e)(f))(d)),(g(k(m)(n))(l)),(o)

Output in width: agobcdklefmn

Output binareThree: (a(b(c(e(f))(d)))(g(k(m(n))(l))(o)))

Тесты не удовлетворяющие условиям ввода

(a,)
(a
b)
(a)(b)
(a),(b)(c)
(a(b(c))
(a(b(c%)))
(a+b)

Вывод: *Были изучены бинарные деревья и леса, реализовано бинарное дерево на векторе. Решение данной задачи на основе вектора нецелесообразно, ввиду сложности и отсутствия преимуществ. Реализация на списках является для данной задачи более простой и понятной.*

Приложение

Код программы lab4.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>

template<typename T>
class BinaryTree {

public:
    struct Element {
        T value;
        int parent;
        int deep;
        int num;
        bool isEnable;
    };
    Element *array;

    void setElementValue(char value, int index) {
        array[index].value = value;
    }


    explicit BinaryTree(std::string &myString, int maxsize) {
        maxSize_ = maxsize;
        array = new Element[maxSize_];
        treeDataProcessing(myString);
    }

    int findParent(int num, int your_deep) {
        for (int i = num; i > -1; --i) {
            if (array[i].deep + 1 == your_deep) {
                return i;
            }
        }
        return -1;
    }

    void treeDataProcessing(std::string &my_string) {

        int length = my_string.length();
        int my_deep = 1;
        int count = 0;
        for (int i = 1; i < length; ++i) {
            if (my_string[i] == '(') {
                my_deep++;
            } else if (my_string[i] == ')') {
                my_deep--;
            } else if (my_string[i] == ',') {

            } else {
                setElementValue(my_string[i], count);
                array[count].deep = my_deep;
```

```
                array[count].num = count;
                array[count].parent = findParent(count, my_deep);
                array[count].isEnable = false;
                count++;
            }

        }
    }

    void setNewDeep(int index) {
        array[index].deep++;
        for (int i = 0; i < maxSize_; ++i) {
            if (array[i].parent == index) {
                setNewDeep(i);
            }
        }
    }

    void createBinaryTree() {
        for (int i = 0; i < maxSize_; ++i) {
            if (!array[i].isEnable) {
                std::vector<int> vector;
                //vector.insert(vector.end(),i);
                int tmpDeep = array[i].deep;
                int tmpParent = array[i].parent;
                for (int j = i; j < maxSize_; ++j) {
                    if (tmpDeep == array[j].deep && tmpParent == array[j].parent &&
!array[j].isEnable) {
                        vector.insert(vector.end(), j);
                        array[j].isEnable = true;
                    }
                }
                while (vector.size() != 1) {

                    int vector_length = vector.size();
                    for (int j = 1; j < vector_length; ++j) {
                        setNewDeep(vector[j]);
                    }
                    //увеличиваем глубину на 1 всем его(детям)
                    array[vector[1]].parent = vector[0];
                    vector.erase(vector.begin());
                }

                //для отладки
                /*std::cout << "!!!!!!!!!!\n";
                printlnTree();
                std::cout << "!!!!!!!!!!\n";*/
            }
        }
    }


    void printlnTree() {
        for (int i = 0; i < maxSize_; ++i) {
            std::cout << array[i].value;
        }
        std::cout << '\n';
        for (int i = 0; i < maxSize_; ++i) {
            std::cout << array[i].deep;
        }
        std::cout << '\n';
```

```
        for (int i = 0; i < maxSize_; ++i) {
            std::cout << array[i].parent;
        }
        std::cout << '\n';
        for (int i = 0; i < maxSize_; ++i) {
            std::cout << array[i].value << " deep " << array[i].deep << " parent " <<
array[i].parent << "\n";
        }
        std::cout << '\n';
    }

    void printInWidth() {
        int max_deep = 1;
        for (int j = 0; j < maxSize_; ++j) {
            if (array[j].deep > max_deep) {
                max_deep = array[j].deep;
            }
        }

        for (int i = 1; i <= max_deep; ++i) {
            for (int j = 0; j < maxSize_; ++j) {
                if (i == array[j].deep) {
                    std::cout << array[j].value;
                }
            }
        }
        std::cout << "\n";
    }

    void printResult(int parent) {

        for (int i = 0; i < maxSize_; ++i) {
            if (array[i].parent == parent) {
                std::cout << "(";
                std::cout << array[i].value;
                printResult(i);
                std::cout << ")";
            }
        }
    }

    ~BinaryTree() {
        delete[] array;
    }

private:
    int maxSize_;
};

bool checkExpression(int &a, const std::string my_string) {
    int length = my_string.length();
    int count = 0;
    int countBrackets = 0;

    if(length==0){
        return false;
    }

    if(my_string[0]!='('&&my_string[length-1]!=')'){
        return false;
    }
```

```
    for (int i = 0; i < length; ++i) {
        if (my_string[i] != '(' && my_string[i] != ')' && my_string[i] != ',') {
            if (!isalpha(my_string[i])) {
                return false;
            }
            count++;
        }
        if (my_string[i] == '(') {
            countBrackets++;
        }
        if (my_string[i] == ')') {
            countBrackets--;
        }
        if (my_string[i] == ',') {
            if(countBrackets!=0){
                return false;
            }
        } else {
            if (countBrackets == 0 && my_string[i+1] != ',' && (i != length - 1)) {
                return false;
            }
        }
        if (countBrackets < 0) {
            return false;
        }
    }
    a = count;
    return countBrackets == 0;
}

int mainCheck(std::string &expression) {

    int countAlpha = 0;
    //std::string expression = "(a(b)(c(e)(f))(d)),(g(k(m)(n))(l)),(o)";
    if (checkExpression(countAlpha, expression)) {
        BinaryTree<char> binaryTree(expression, countAlpha);
        //разкомент для детального вывода
        //binaryTree.printlnTree();
        //std::cout << "\n";
        std::cout << "Output in width: ";
        binaryTree.printInWidth();
        binaryTree.createBinaryTree();
        //binaryTree.printlnTree();
        std::cout << "Output binareThree: ";
        binaryTree.printResult(-1);
        std::cout << "\n";
    } else {
        std::cout << "not correct. String is incorrect" << "\n";
    }


    return 0;
}

int main() {

    int your_choose = 0;

    std::cout << "If you want to enter data from a file, enter \'1\'\n";
    std::cout << "If you want to enter data manually, enter \'2\'\n";
```

```
    std::cin >> your_choose;

    if (your_choose == 1) {
        std::ifstream fin;
        fin.open("C:\\Users\\Alex\\Desktop\\test4.txt");

        if (fin.is_open()) {
            std::cout << "Reading from file:" << "\n";

            int super_count = 0;

            while (!fin.eof()) {

                super_count++;

                std::string str;
                getline(fin, str);

                std::cout << "test #" << super_count << " \"" + str + "\"" << "\n";
                mainCheck(str);

            }
        } else {
            std::cout << "File not opened";
        }

        fin.close();
    } else {
        if (your_choose == 2) {
            std::cout << "Enter data \n";
            std::string str;
            std::cin >> str;
            mainCheck(str);
        }
    }

    return 0;
}
```